Data Engineering – Project 4: Spatial Data

April 24, 2025

1 Introduction

Welcome to the fourth project, which is about spatial data.

As usual, the exercises require that you load a dataset and process it as required, saving the indicated file. Please remember that the provided example datasets and configuration may be different from what is used to test your programs, but will always follow the guidelines specified in the exercises.

2 Exercises

The file `proj4_params.json` contains a JSON dictionary with parameters which will be used throughout the exercises. Load it.

2.1 Exercise 1: Loading data and basic operations

4 points (2 for each file)

Load a set of points from a GeoJSON file called `proj4_points.geojson`. The points will always be located somewhere in Poland. The CRS will be specified in the GeoJSON file according to the specification and should be handled by GeoPandas correctly without any extra work.

Each point has a unique identifier in a columns specified by the `id_column` parameter loaded from the JSON file above (in the example dataset, it is `lamp_id`).

For each point, count the number of points (including the point itself) that are within 100 metres of that point. For instance, if the points represents street lamps, count the number of lamps that are within 100 metres of each lamp (including the lamp itself).

Note that while it would *theoretically* be possible to iterate over the dataset and find neighbours of each point separately, such approach would be very *inefficient*.

Therefore, for this task, you should perform a spatial join, using the appropriate binary predicate.

You can find the predicates available for a given GeoDataFrame as follows:

```
>>> gdf.sindex.valid_query_predicates
{None, 'contains', 'contains_properly', 'covered_by', 'covers', 'crosses',
 'intersects', 'overlaps', 'touches', 'within'}
```

Note that there is no predicate for distance-based spatial relationships. Therefore, a common approach is to (temporarily) create a buffer for one of the datasets being joined and use a predicate such as `intersects`.

Save the results to a file called `proj4_ex01_counts.csv`, with two columns:

- the identifier column, with its original name,
- a column called `count` with the number of “neighbouring” points.

An example file could look like this:

```
lamp_id,count
5907,16
5908,16
5909,17
5910,20
5911,9
(...)
```

Now save the *latitude* and *longitude* of all points to a CSV file called `proj4_ex01_coords.csv`, with the following columns:

- the identifier column, with its original name,
- `lat` for latitude,
- `lon` for longitude.

Keep exactly 7 decimal digits for the coordinates (trailing zeroes can be skipped).

An example file could look like this:

```
lamp_id,lat,lon
5907,50.0740434,19.8991355
5908,50.0750528,19.8913931
5909,50.0730553,19.8982101
(...)
```

2.2 Exercise 2: Loading data from OpenStreetMap

4 points

The `city` parameter contains the name of the city where the points are located (e.g. *Kraków, Poland*). That city will be identifiable by OSMnx.

Load OpenStreetMap data for that city into a GeoDataFrame using OSMnx. Only include roads which can be used by motor vehicles, and from those, only include *tertiary* ones, e.g. those with the highway key set to `tertiary`.

Structure your GeoDataFrame so that it contains the following columns:

- `osm_id` – the OpenStreetMap identifier of the street,
- `name` – the name of the street,
- `geometry` – the geometry.

Save it to `proj4_ex02_roads.geojson`.

2.3 Exercise 3: Spatial joins

3 points

For each of the roads obtained in Exercise 2, count the number of points, loaded in Exercise 1, that are within 50 metres of the line representing the axis of the road.

Minimize the risk that one lamp will be "captured" by more than one street by applying appropriate *endcaps*. Please refer to the documentation of the buffer function to find the correct values.

Save the results to a CSV file called `proj4_ex03_streets_points.csv`, with the following columns:

- `name`, with the name of the street,
- `point_count`, with the number of points within 50 metres of that street.

Only include streets where more than zero points have been located. If there are multiple OSM *ways* with the same street name, aggregate them. An example file could look as follows:

```
name,point_count
Aleja 3 Maja,209
Aleja Kijowska,191
Aleja Marszałka Ferdinanda Focha,87
(...)
```

2.4 Exercise 4: Drawing maps

5 points (2 for GDF, 3 for images)

The file `proj4_countries.geojson` contains polygons with boundaries of several countries from all over the World. Load the GeoJSON file into a GeoDataFrame.

Each feature in the file has a property called `name`, which contains the name of the country.

Modify the GeoDataFrame so that:

- the boundaries are *hollow*, not *filled*,
- the horizontal/vertical aspect ratios (proportions) of the shapes are correct.

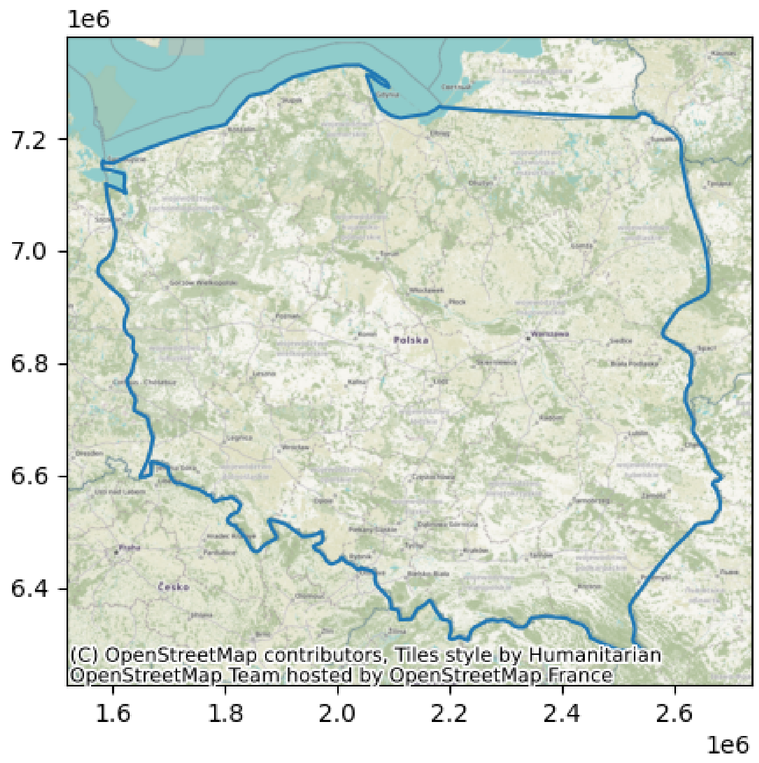
Please note that while we want the shapes to not be "squashed" either horizontally or vertically, it is acceptable for them to be have distortions which result from using Mercator projections. In other words, the shapes should look "good" on the map.

Save the modified GeoDataFrame to `proj4_ex04_gdf.pkl`.

Now render the boundary of each country to a separate PNG file, adding a background map to provide context.

The name of the file should follow the scheme: `proj4_ex04_COUNTRY.png`, where `COUNTRY` is the country name in lowercase, e.g. `proj4_ex04_poland.png`, `proj4_ex04_italy.png`, etc.

An example rendering could look like this:


1e6
7.2
7.0
6.8
6.6
6.4
(C) OpenStreetMap contributors, Tiles style by Humanitarian OpenStreetMap Team hosted by OpenStreetMap France
1.6
1.8
2.0
2.2
2.4
2.6
1e6

3 Submit your solution

As usual, commit your program to your GitLab project repository.

Save it as `project04/project04.py`.